Agents that **Reduce Work and Information Overload**

Computers are becoming the vehicle for an increasing range of everyday activities. Acquisition of news and information, mail and even social interactions and entertainment have become more and more computer based. At the same time, an increasing number of untrained users are interacting with computers, and this number will continue to rise as technologies such as hand-held computers and interactive television become popular.

Unfortunately, these technological developments are not in line with a change in the way people interact with computers. The currently dominant interaction metaphor of *direct manipulation* [21] requires the user to initiate all tasks explicitly and to monitor all events. This metaphor will have to change if untrained users are to make effective use of the computer and networks of tomorrow.

Techniques from the field of AI, in particular so-called "autonomous agents," can be used to implement a complementary style of interaction, which has been referred to as *indirect management* [9]. Instead of user-initiated interaction via commands and/or direct manipulation, the user is engaged in a cooperative process in which human and computer agents both initiate communication, monitor events and perform tasks. The metaphor used is that of a *personal assistant* who is *collaborating with the user* in the same work environment. The assistant becomes gradually more effective as it learns the user's interests, habits and preferences (as well as those of his or her community). Notice that the agent is not necessarily an interface between the computer and the user. In fact, the most successful interface agents are those that do not prohibit the user from taking actions and fulfilling tasks personally (see Figure 1).

Agents assist users in a range of different ways: they hide the complexity of difficult tasks, they perform tasks on the user's behalf, they can train or teach the user, they help different users collaborate, and they monitor events and procedures.

The set of tasks or applications an agent can assist in is virtually unlimited: information filtering, information retrieval, mail management, meeting scheduling, selection of books, movies, music, and so forth.

This article focuses on a novel approach to building interface agents. It presents results from several prototype agents that have been built using this approach, including agents that provide personalized assistance with meeting scheduling, email handling, electronic news filtering and selection of entertainment.

Approaches to Building Agents

The idea of employing agents in the interface to delegate certain computer-based tasks was introduced by visionaries such as Nicholas Negroponte [19] and Alan Kay [8]. More recently, several computer manufacturers have adopted this idea to illustrate their vision of the interface of the future (cf., videos produced in 1990-1991 by Apple, Hewlett Packard, Digital and the Japanese FRIEND21 project). Even though a great amount of work has gone into the modeling and construction of agents, currently available techniques are still far from being able to produce the high-level, humanlike interactions depicted in these videos. Two main problems have to be solved when building software agents. The first problem is that of *competence:* how does an agent acquire the knowledge it needs to decide when to help the user, what to help the user with and how to help the user? The second problem is that of *trust:* how can we guarantee the user feels comfortable delegating tasks to an agent? Two previous approaches for build-

ing interface agents can be distinguished. Neither one provides a satisfactory solution to these problems.

The first approach consists in making the end-user program the interface agent. Malone and Lai's Oval (formerly Object-Lens) system [12] has "semi-autonomous agents" which consist of a collection of user-programmed rules for processing information related to a particular task. For example, the Oval user can create an email sorting agent by creating a number of rules that process incoming mail messages and sort them into different folders. Once created, these rules perform tasks for the user without having to be explicitly invoked by the user. Similarly, one can buy "agents" that can be programmed by the user to provide information filtering services (e.g., selecting any new article that mentions MIT Media Lab).

The main problem of this approach is that it does not deal with the competence criterion in a satisfactory way. The approach requires too much insight, understanding and effort from the end-user, since the user has to recognize the opportunity for employing an agent, take the initiative to create an agent, endow the agent with explicit knowledge (specifying this knowledge in an abstract language), and maintain the agent's rules over time (as work habits or interests change).

Trusting the agent is less of a problem in this approach, provided that the user trusts his or her own programming skills. However, the programs we write typically behave differently than expected, even when we trust our programming skills.

The second approach, called the "knowledge-based approach," consists in endowing an interface agent with extensive domain-specific background knowledge about the application and the user (called a domain model and user model, respectively). This approach is adopted by the majority of people working in AI on intelligent user interfaces [25]. At runtime, the interface agent uses its knowledge to recognize the user's plans and to find opportunities for contributing to them. For example, UCEgo [1] is an interface agent designed to help a user solve problems in using the Unix operating system. The UCEgo agent has a large knowledge base about how to use Unix, incorporates goals and meta-goals, and does planning (e.g., UCEgo can volunteer information or correct the user's misconceptions).

Both competence and trust constitute problems in the knowledge-based approach. The first problem related to competence is that the approach requires a huge amount of work from the knowledge engineer. A large amount of application-specific and domain-specific knowledge has to be entered into the agent's knowledge base. Little of this knowledge or the agent's control architecture can be used when building agents for other applications. The second problem is that the knowledge of the agent is fixed once and for all. It cannot be customized to individual user habits and preferences. The possibility of providing an agent with all the knowledge it needs to always comprehend the user's sometimes unpredictable actions is questionable.

In addition to the competence problem, there is also a problem with trust. It is probably not a good idea to give a user an interface agent that is very sophisticated, qualified and autonomous from the start. Schneiderman has argued convincingly that such an agent would leave the user with a feeling of loss of control and understanding [18]. Since the agent has been programmed by someone else, the user may not have a good model of the agent's limitations, the way it works, and so forth.

Training a Personal Digital Assistant

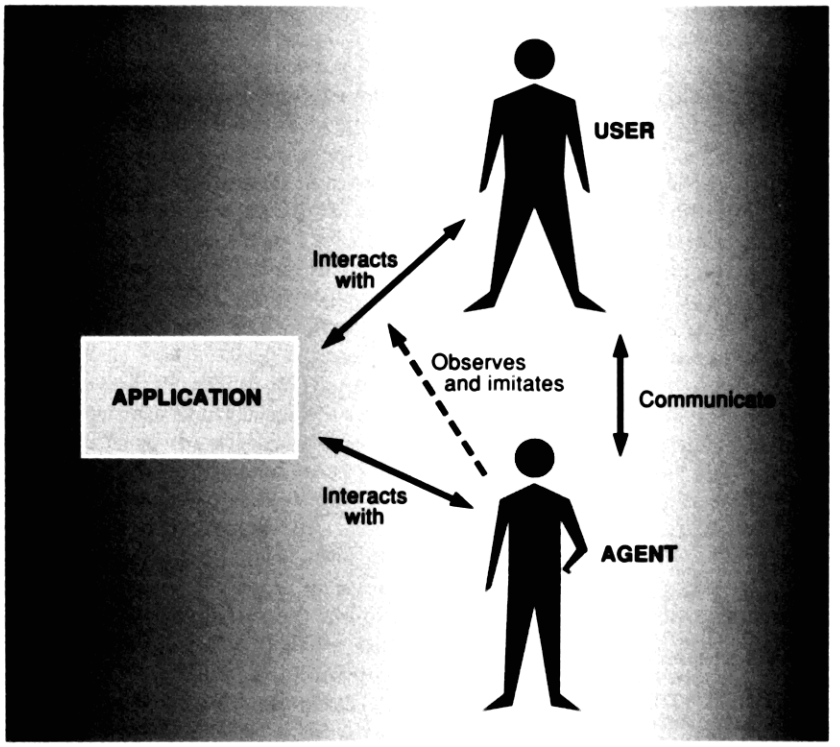
In our work, we explore an alternative approach to building interface agents that relies on machine learning techniques. A similar approach has been reported upon by Dent *et al.* [4]. The hypothesis tested is that, under certain conditions, an interface agent can "program itself" (i.e., it can acquire the knowledge it needs to assist its user). The agent is given a minimum of background knowledge, and it learns appropriate "behavior" from the user and from other agents. The particular conditions that have to be fulfilled are: (1) the use of the application has to involve a substantial amount of repetitive behavior (within the actions of one user or among users), and (2) this repetitive behavior is potentially different for different users. If the latter condition is not met (i.e., the repetitive behavior demonstrated by different users is the same), a knowledge-based approach might prove to yield results faster than a learning approach. If the former condition is not met, a learning agent will not be able to learn anything (because there are no regularities to learn in the user's actions).

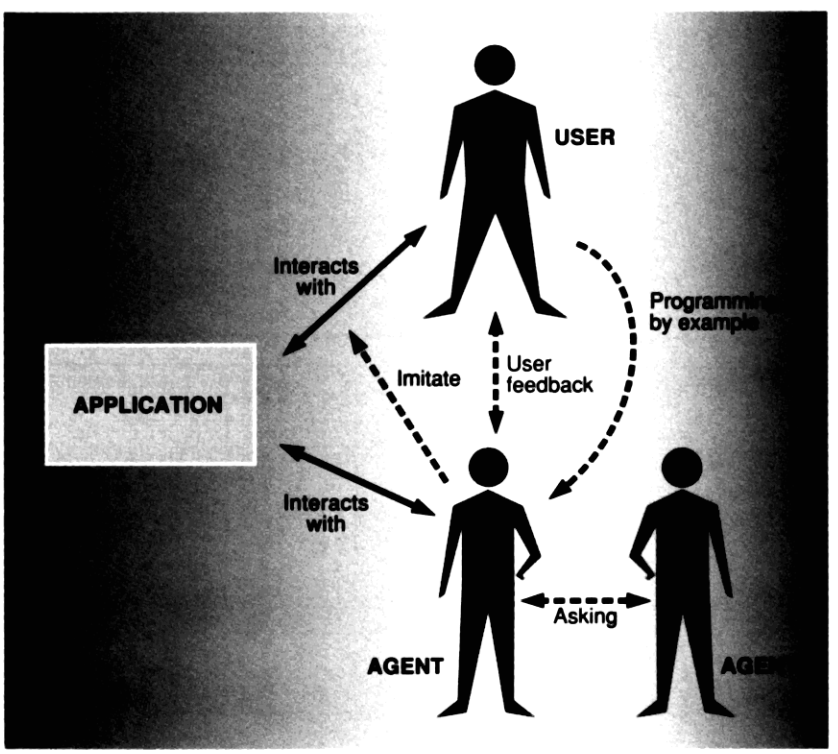
The machine learning approach is inspired by the metaphor of a personal assistant. Initially, a personal assistant is not very familiar with the habits and preferences of his or her employer and may not even be very helpful. The assistant needs some time to become familiar with the particular work methods of the employer and organization at hand. However, with every experience the assistant learns, either by watching how the employer performs tasks, by receiving instructions from the employer, or by learning from other more experienced assistants within the organization. Gradually, more tasks that were initially performed directly by the employer can be taken care of by the assistant.

The goal of our research is to demonstrate that a learning interface agent can, in a similar way, become gradually more helpful and competent. In addition, we attempt to demonstrate that the learning approach also presents a satisfactory solution to the trust problem. If the agent gradually develops its abilities—as is the case in our approach—the user is also given time to gradually build up a model of how the agent makes decisions, which is one of the prerequisites for a trust relationship. Furthermore, the particular learning approach adopted allows the agent to give "explanations" for its reasoning and behavior in a language the user is familiar with, namely in terms of past examples similar to the current situation. For example, "I thought you

Figure 1. The interface agent does not act as interface or layer between the user and the application. It behaves as a personal assistant that cooperates with the user on the task. The user is able to bypass the agent.

Figure 2. The interface agent learns in four different ways: (1) it observes and imitates the user's behavior, (2) it adapts based on user feedback, (3) it can be trained by the user on the basis of examples, and (4) it can ask for advice from other agents assisting other users.





might want to take this action because this situation is similar to this other situation we have experienced before, in which you also took this action" or "because assistant *Y* to person *Z* also performs tasks that way, and you and *Z* seem to share work habits."

Finally, we believe the learning approach has several advantages over the previous two approaches. First, it requires less work from the end-user and application developer. Second, the agent can more easily adapt to the user over time and become customized to individual and organizational preferences and habits. Finally, the approach helps in transferring information, habits and know-how among the different users of a community. The results described in a later section support all of these hypotheses and predictions.

A learning agent acquires its competence from four different sources (see Figure 2). First, the interface agent learns by continuously "looking over the shoulder" of the user as the user is performing actions. The interface agent can monitor the activities of the user, keep track of all of his or her actions over long periods of time (weeks or months), find regularities and recurrent patterns and offer to automate these. For example, if an email agent notices that a user almost always stores messages sent to the mailing list "intelligent-interfaces" in the folder `pattie:email:int-int.txt`, then it can offer to automate this action the next time a message sent to that mailing list is read. Similarly, if a news filtering agent detects some patterns in the articles the user reads, then it can offer similar articles to the user when it discovers them.

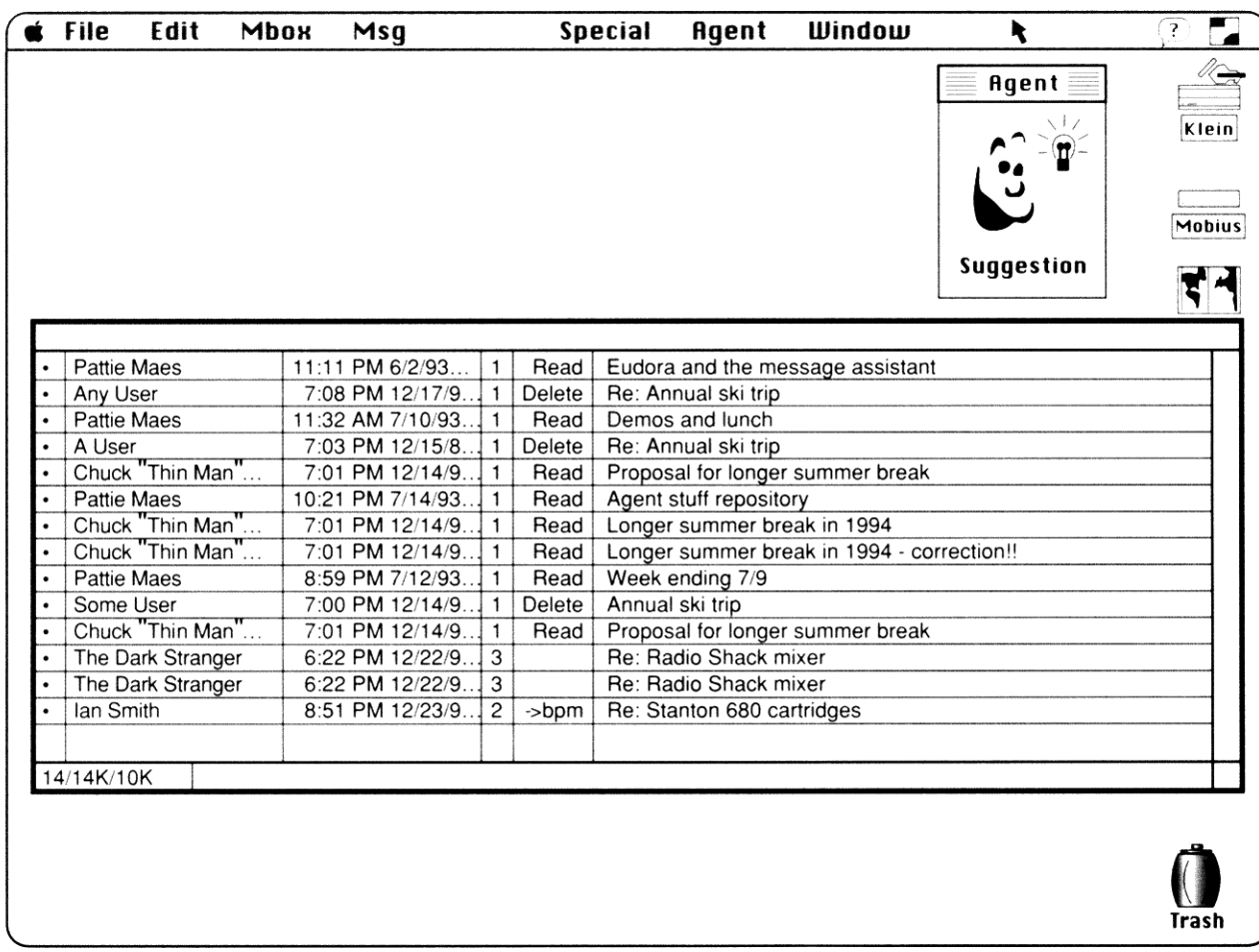
A second source for learning is direct and indirect user feedback. Indirect feedback happens when the user neglects the suggestion of the agent and takes a different action instead. This can be as subtle as the user changing the order in which he or she reads incoming messages, not reading some articles suggested by the agent, or reading articles not sug-

gested by the agent. The user can also give explicit negative feedback for actions automated by the agent ("don't do this again" or "I dislike this article").

Third, the agent can learn from examples given explicitly by the user. The user can train the agent by giving it hypothetical examples of events and situations and telling the agent what to do in those cases. The interface agent records the actions, tracks relationships among objects, and changes its example base to incorporate the example that it is shown. For instance, the user can teach a mail clerk agent to save all messages sent by a particular person in a particular folder by creating a hypothetical example of an email message (which has all aspects unspecified except for the sender field) and dragging this message to the chosen folder. Similarly, the user can instruct a news filtering agent by giving it examples of (partially specified) articles (e.g., "select any article in which the word MIT appears").

Finally, a fourth method used by the interface agent to acquire competence is to ask for advice from agents that assist other users with the same task (and that may have built up more experience). If an agent does not know itself what action is appropriate in a certain situation, it can present the situation to other agents and ask "what action they recommend for that situation." For example, if an email message arrives that has been sent by Nicholas Negroponte, then the email agent can ask other agents what to do with that message. If a majority of the other agents recommends that the message has high priority and should be presented to the user for reading right away, then the agent can offer this recommendation to its user, even though the agent never previously observed the user deal with messages from Nicholas Negroponte. Rather than averaging the recommendations of all other agents in the community, a user might also inform his or her agent to accept suggestions from one or more specific agents which assist specific users. For example, if one person in the lab is an expert in the use of a particular piece of software, then other users can instruct their agents to accept advice about that software from the agent of that expert user.

Additionally, the agent can learn from experience which agents are good sources for suggestions. It can learn to trust agents that in the past



•	Pattie Maes	11:11 PM 6/2/93...	1	Read	Eudora and the message assistant
•	Any User	7:08 PM 12/17/9...	1	Delete	Re: Annual ski trip
•	Pattie Maes	11:32 AM 7/10/93...	1	Read	Demos and lunch
•	A User	7:03 PM 12/15/8...	1	Delete	Re: Annual ski trip
•	Chuck "Thin Man"...	7:01 PM 12/14/9...	1	Read	Proposal for longer summer break
•	Pattie Maes	10:21 PM 7/14/93...	1	Read	Agent stuff repository
•	Chuck "Thin Man"...	7:01 PM 12/14/9...	1	Read	Longer summer break in 1994
•	Chuck "Thin Man"...	7:01 PM 12/14/9...	1	Read	Longer summer break in 1994 - correction!!
•	Pattie Maes	8:59 PM 7/12/93...	1	Read	Week ending 7/9
•	Some User	7:00 PM 12/14/9...	1	Delete	Annual ski trip
•	Chuck "Thin Man"...	7:01 PM 12/14/9...	1	Read	Proposal for longer summer break
•	The Dark Stranger	6:22 PM 12/22/9...	3		Re: Radio Shack mixer
•	The Dark Stranger	6:22 PM 12/22/9...	3		Re: Radio Shack mixer
•	Ian Smith	8:51 PM 12/23/9...	2	->bpm	Re: Stanton 680 cartridges
14/14K/10K					

Figure 3. The email agent makes recommendations to the user (middle column). It predicts what actions the user will perform on messages, such as which messages will be read and in which order, which messages will be deleted, forwarded, archived, and so on.

have proven to recommend actions the user appreciated. The entertainment selection agent discussed next uses this technique to learn which other users have entertainment tastes similar to its user's tastes, and thus are good sources of suggestions.

Some Examples of Existing Agents

Four agents have been built using the learning approach previously discussed:

- An agent for electronic mail handling
- An agent for meeting scheduling
- An agent for electronic news filtering (Usenet Netnews).
- An agent that recommends books, music or other forms of entertainment

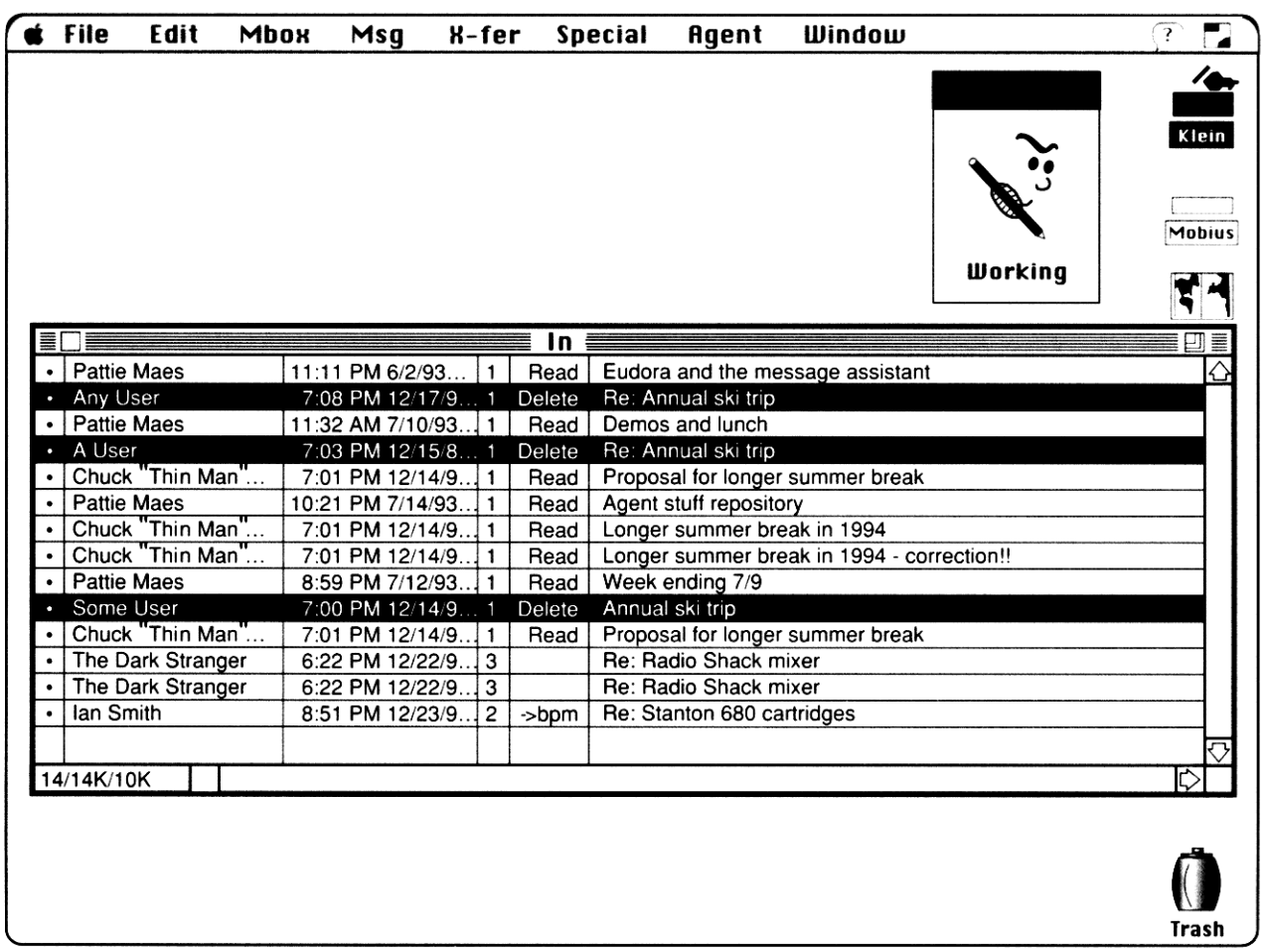
The choice of these domains was motivated by our dissatisfaction with the ways these tasks are currently handled. Many valuable hours are wasted dealing with junk mail, scheduling and rescheduling meetings, searching for relevant information among heaps of irrelevant information, and browsing through lists of books, music, and television programs in search of something interesting.

Electronic Mail Agent

Maxims [13] is an agent that assists the user with email. Maxims learns to prioritize, delete, forward, sort and archive mail messages on behalf of the user (Figures 3 and 4). Maxims is implemented in Macintosh Common Lisp. It communicates with the commercial email package Eudora [6] using Apple Events. The main learning technique used by Maxims is memory-based reasoning [24]. The agent continuously "looks over the shoulder" of the user as the user deals with email. As the user performs actions, the agent memorizes all of the situation-action pairs generated. For example, if the user saves a particular message after having read it, the mail agent adds a description of this situation and the action taken by the user to its memory of examples. Situations are described in terms of a set of features, which are currently handcoded. In this domain, the agent keeps track of the sender and receiver of a message, the Cc: list, the keywords in the Subject: line, whether the message has been read or not, whether it is a reply to a previous message, and so on.

When a new situation occurs that can be due to the user taking an action or due to some external event such as a message arriving, the agent will try to predict the action(s) of the user, based on the examples stored in its memory. The agent compares the

Figure 4. The user can select some of the suggestions made by the agent and ask the agent to execute them.

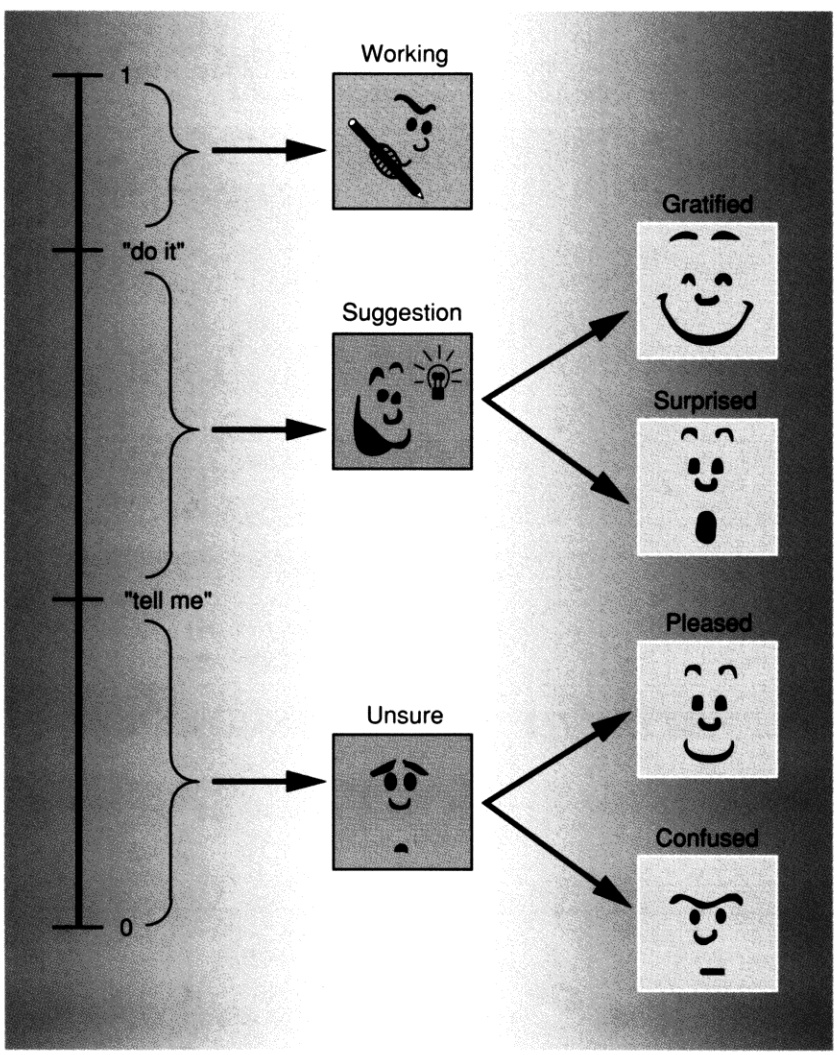


new situation with the memorized situations and tries to find a set of nearest neighbors (or close matches). The most similar of these memorized situations contribute to the decision of which action to take or suggest in the current situation. The distance metric used is a weighted sum of the differences for the features that make up a situation. Some features carry more weight than others. The weight of a feature is determined by the agent. Occasionally (e.g., at night), the agent analyzes its memory and determines the correlations between features and actions taken. For example, the agent may detect that the "from" field of an email message is highly correlated to whether its user reads the message, while the "date" field is not correlated. The detected correlations are employed as weights in the distance metric.

The agent does not only predict which action is appropriate for the current situation. It also measures its confidence in each prediction. The confidence level is determined by whether or not all the nearest neighbors recommended the same action, how close/distant the nearest neighbors are, and how many examples the agent has memorized (a measure of the accuracy of the correlation weights).

Two thresholds determine how the agent uses its prediction. When the confidence level is above the "do-it" threshold, then the agent autonomously takes the action on behalf of the user. In that case, it writes a report for the user about the action it automated. The user can ask the agent for its report of automated actions any time. If the confidence level is above the "tell-me" threshold, then the agent will offer its suggestion to the user, but will wait for the user's confirmation to automate the action. The user is responsible for setting the "tell-me" and "do-it" thresholds for actions at levels where the user feels comfortable. For example, if the user feels paranoid about the agent autonomously deleting messages, then the user can set the "do-it" threshold for that action at a maximum.

Figure 5. Simple caricatures convey the state of the agent to the user.



The agent communicates its internal state to the user via facial expressions (see Figure 5). These appear in a small window on the user's screen. The faces have a functional purpose: they make it possible for the user to get an update on what the agent is doing "in the blink of an eye." There are faces for "thinking" (the agent is comparing the current situation to memorized situations), "working" (the agent is automating an action), "suggestion" (the agent has a suggestion), "unsure" (the agent does not have enough confidence in its suggestion), and so forth. The "pleased" and "confused" face help the user gain information about the competence of the agent (if the agent never offers its suggestion, but it always shows a pleased face after the user takes an action, then clearly the "tell-me" threshold should be lowered. The agents have deliberately all been drawn as simple cartoon faces, in order not to encourage unwarranted attribution of human-level intelligence.

The Maxims agent gradually gains competence by observing the user and acquiring more examples. In order to deal with this slow start problem, two additional competence acquisition schemes exist. First of all, it is possible for the user to instruct the agent explicitly. If the user does not want to wait for the agent to pick up a certain pattern, then the user can create a hypothetical situation and show the agent what should be done. This functionality is imple-

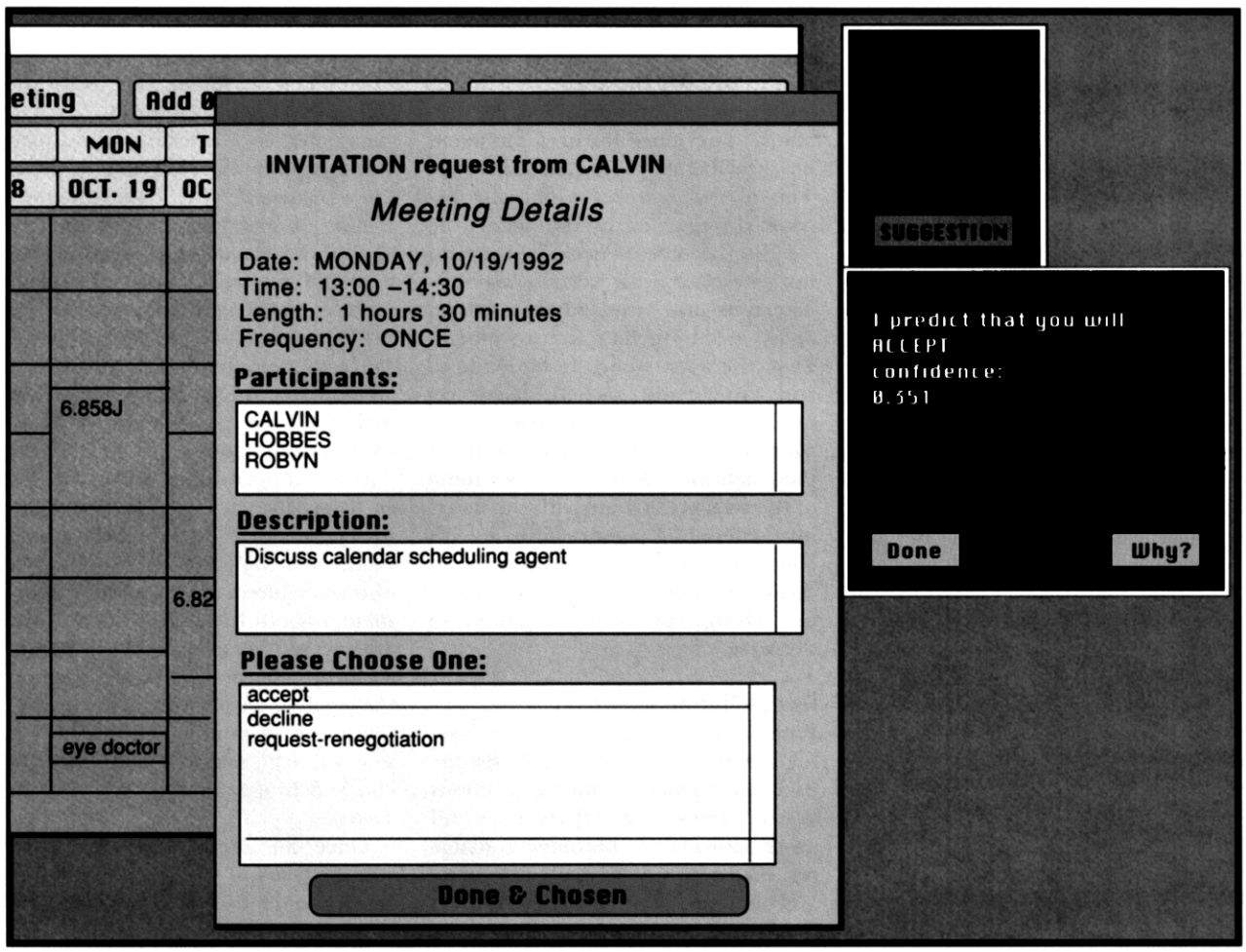


Figure 6. The calendar agent offers its suggestion to the user. The user can accept, take a different action or ask for an explanation.

mented by adding the example to the agent's memory, including "wildcards" for the features not specified in the hypothetical situation. For instance, the user can create a hypothetical message from Negroponte and show the agent that message has high priority. The new situation-action pair will match all situations in which an email message has been received from Negroponte. By varying the way in which such hypothetical examples are treated when selecting an action and when compiling statistics, both *default* and *hard-and-fast rules* can be implemented within the memory-based learning framework [11].

A second method that allows the agent to start from more than scratch is multiagent collaboration. When the agent does not have enough confidence in its prediction (confidence that is less than the "tell-me" threshold), it asks for help from other agents assisting other users with email. The agent sends part of the description of the situation to other agents via email and awaits their response. For example, if a new Media Lab user/agent pair receives a message from Negroponte, then that agent will ask other agents what it should do with that message. Some other agents will recommend that the message is important and should be immediately presented to the user for reading. The agent will make a prediction based on the different suggestions that are returned. The agent gradually learns which other agents are trustworthy sources of information for certain classes of problems. Every agent models how much it trusts other agents' advice. The trust level is increased or decreased when the action eventually taken by the user is compared with the recommendations (and confidence levels) of peer agents. The multiagent communication is an excellent method for transfer of information and competence among different users in a workgroup.

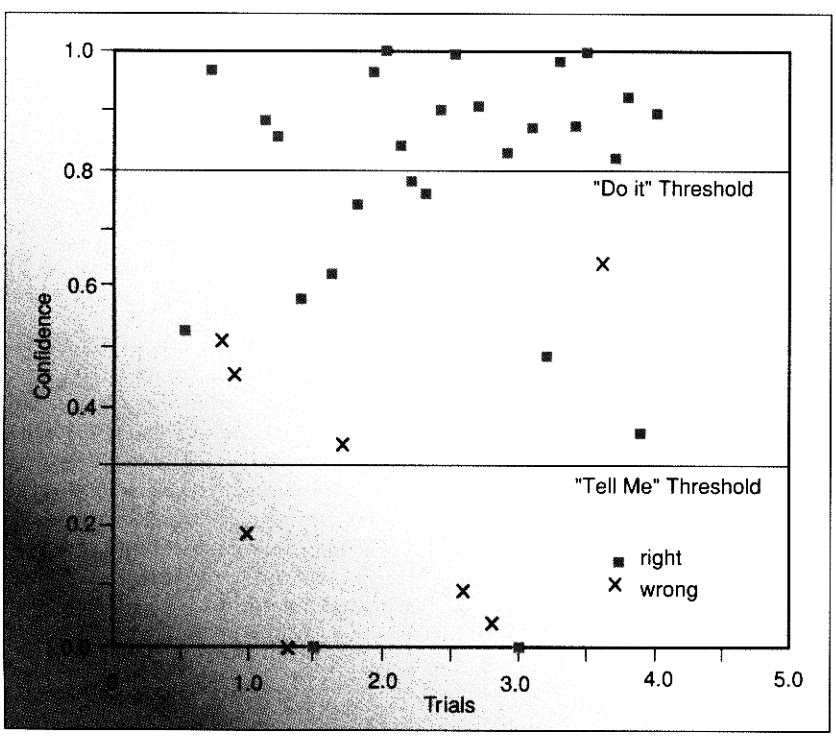
Meeting Scheduling Agent

The learning agent just described is generic. It can be attached to any application, provided the application is scriptable and recordable. The same agent was attached to a meeting scheduling software package [10, 11]. The resulting agent assists a user with the scheduling of meetings (accept/reject, schedule, reschedule, negotiate meeting times, etc.). The meetings scheduling agent is again implemented in MCL (see Figure 6). Meeting scheduling is another exam-

ple of a task that fulfills the criteria for learning interface agents: the behavior of users is repetitive, but nevertheless very different for individual users. Some people prefer meetings in the morning, others in the afternoon. Some group meetings, others spread them out. Different people have different criteria for which meetings are important, which meeting initiators are important (and should be accommodated), and so on. The learning interface agent approach is ideally suited for assisting the user in a very personalized way by automating the scheduling task according to the unique habits of the user.

Both the Maxims agent and the meeting scheduling agent have been tested by real users. The results of these user tests are very encouraging. Users are eager to try out interface agents. They welcome whatever help they can get with their work overload. Users reported they felt comfortable delegating tasks to the agents. The tests revealed it is important to provide the agent with an extensive set of features to describe situations. The more features the agent has, the better the agent performs. The useless features eventually become disregarded by the agent (the weights become 0 because they do not correlate with certain actions). The tests also revealed that several areas need further improvement. First, the agents have to be made to run faster, and second, users requested they be able to instruct the agent to forget or disregard some of their behavior. Figure 7 shows some of the results obtained with the meeting scheduling agent. The agent's confidence in correct predictions increases with time, while its confidence in wrong predictions tends to decrease.

Figure 7. Typical results from a learning meeting scheduling agent. The confidence level in correct predictions increases with time, while the confidence level in wrong predictions tends to decrease.



News Filtering Agent

Probably one of the more widely useful agents is an agent that helps the user select articles from a continuous stream of news [22, 23]. As more and more information becomes available on the network (see World Wide Web, on-line news feeds, etc.), users become more and more desperate for tools that will help them filter this stream of information and find articles of interest to them.

NewT is a system that helps the user filter Usenet Netnews. It is implemented in C++ on a Unix platform. A user can create one or many "news agents" and train them by means of examples of articles that should or should not be selected. For example, Figure 8 shows four agents (and icons) created by a user: one for business news, one for political news, one for computer news and one for sports news. An agent is initialized by giving it some positive and negative examples of articles to be retrieved. The agent performs a full text analysis (using the vector-space model [20] for documents) to retrieve the words in the text that may be relevant. It also remembers the structured information about the article, such as the author, source, assigned indices, and so forth. The user can also program the agent explicitly and fill out a set of templates of articles that should be selected (e.g., select all articles by Michael Schrage in the *Los Angeles Times*).

Once an agent has been bootstrapped, it will start recommending articles to the user. The user can give it positive or negative feedback for articles or portions of articles recommended. For example, the user can highlight a word or paragraph and give selective positive or negative feedback. The user can also select the author or source and give positive and negative feedback.

This will increase or decrease the probability that the agent will recommend similar articles in the future. The current implementation only performs content filtering, in the sense that the agent tries to correlate the positive and negative feedback with the contents of the article. It does not perform "social filtering," which would mean agents of different users would exchange recommendations with one another. However, a weaker form of social filtering is implemented in the NewT system: a user can train an agent for a while and then create a duplicate of that agent and give it to another user.

The NewT agents were tested by a group of 12 users. A larger group is

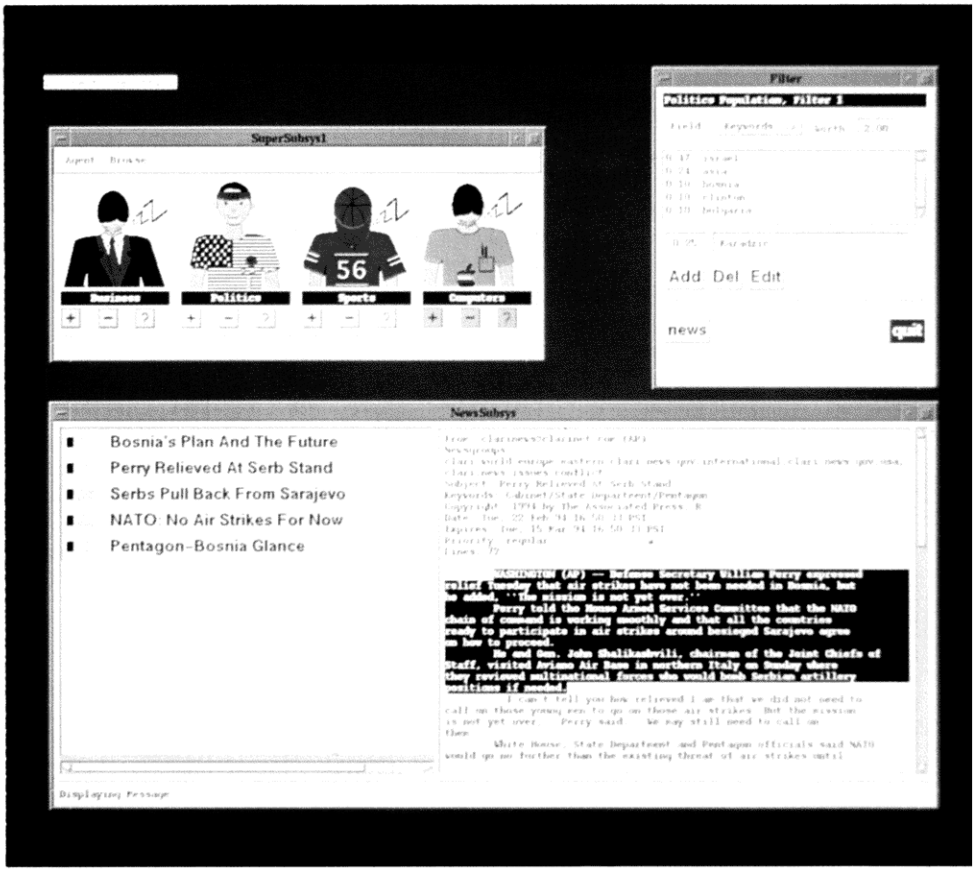


Figure 8. The NewT personalized news filtering system. A user can create a set of agents that assist the user with the filtering of an on-line news source. The agents are trained by means of positive and negative examples of articles to be selected or not selected, respectively.

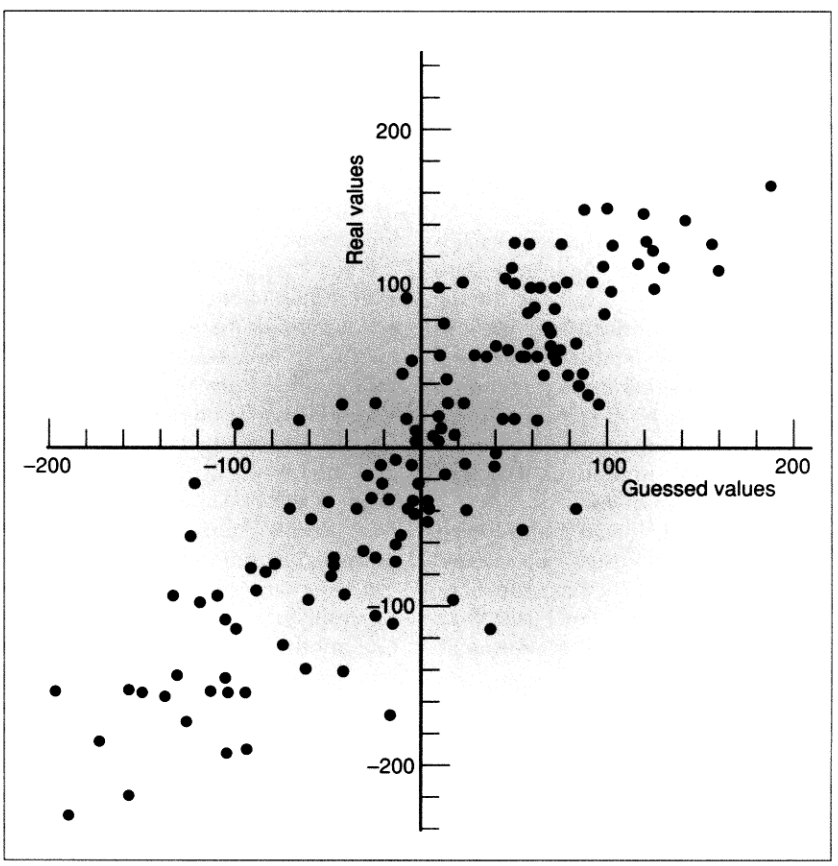


Figure 9. Results obtained with the entertainment selection agents. The Y-axis represents actual values assigned by users to items. The X-axis represents values guessed by the agents based on correlations among users.

using the system on a daily basis. As is the case with the other agent systems previously described, the NewT agents are not meant to automate all of the user's news interests. The agents are able to recommend articles to the user that concern topics (or authors or sources) in which the user has shown a continued interest. The user is still responsible for browsing the news sources to find less predictably interesting articles. Once such articles have been discovered, the user can train the agent to select those kinds of articles in the future. Again, the results obtained with the news agents were very promising. Users liked using the system and found it very useful. The main limitation of the system is its restriction to keywords only. However, if a method for deeper semantic analysis of texts becomes available (e.g., as a result of natural language understanding research), this deeper representation can be learned by using the same statistical learning techniques as are currently used for relevant keywords.

Entertainment Selection Agent

A fourth and final application area is entertainment selection. Of all four applications discussed, this one might have the best potential to become the next "killer application." Currently, critics publish reviews and recommendations which are meant for a large, general audience, but no individualized mechanisms exist to help people select movies, books, television and radio shows based on their personal tastes. However, as soon as

entertainment becomes more interactive, agents can offer personalized, "readership of one" recommendations and critiques. Ringo is a personalized music recommendation system implemented on Unix platform in Perl. A similar system was built for recommending science fiction books [7]. The agents in these system use "social filtering." They do not attempt to correlate the user's interests with the contents of the items recommended. Instead, they rely solely on correlations between different users.

In these systems, every user has an agent that memorizes which books or music albums its user has evaluated, and how much the user liked them. Then, agents compare themselves with other agents. An agent finds other agents that are correlated, that is, agents that have values for similar items and whose values are positively correlated to the values of this agent. Agents accept recommendations from other correlated agents. Basically, what this means is that, if user A and user B have related musical tastes, and A has evaluated an album positively which B has not yet evaluated, then that album is recommended to user B. The actual algorithm is slightly more complex in the sense that agents combine the recommendations from a collection of related agents, rather than a single related agent. Figure 9 illustrates some of the results obtained.

One problem with this approach is how to bootstrap the whole system, so that enough data is available for the agents to start noticing correlations and make recommendations. A second problem is that users can end up relying too much on the recommendation system, and may not enter into the system any new items that they discovered themselves.

In order to deal with both of these problems, "virtual users" are created that represent a particular taste, (e.g., a virtual "Madonna fan" user, who has high ratings for all Madonna albums and no other ratings, or a virtual "cyberspace fan" user, who rates all books on cyberspace highly and has no other ratings). Similarly, a virtual user can be created for a publishing company, like the "MIT Press fan" who rates all MIT Press books highly. By entering such virtual user data into the system, the agent system can bootstrap itself, and agents for actual users can correlate themselves with virtual users.

Discussion

As computers are used for more tasks and become integrated with more services, users will need help dealing with the information and work overload. Interface agents radically change the style of human-computer interaction. The user delegates a range of tasks to personalized agents that can act on the user's behalf. We have modeled an interface agent after the metaphor of a personal assistant. The agent gradually learns how to better assist the user by:

- Observing and imitating the user
- Receiving positive and negative feedback from the user
- Receiving explicit instructions from the user
- Asking other agents for advice

These agents have been shown to tackle two of the hardest problems involved in building interface agents. The agents are *competent:* they become more helpful, as they accumulate knowledge about how the user handles certain situations. They can be *trusted:* the user is able to gradually and incrementally build up a model of the agent's competencies and limitations.

Even though the results obtained with this first generation of agents are encouraging, many open questions for future research remain. Some of these are user interface issues: Should there be one or many agents? Should agents use facial expressions and other means of personification? What is the best metaphor for interface agents? Other questions are more algorithmic and technical: How can we guarantee the user's privacy, especially if agents communicate with one another about their users? How can heterogeneous agents, built by different developers and using different techniques, collaborate? How can a system of incentives be devised, so that users are motivated to share the knowledge their experienced agents have learned? Most importantly, from a legal standpoint, should a user be held responsible for his or her agent's actions and transactions?

Acknowledgments

Beerud Sheth, Robyn Kozierok, Yezdi Lashkari, Max Metral, Carl Feynman, Upendra Shardanand, Cecile Pham and Henry Lieberman were responsible for many of the concepts and for the implementation of the four prototype agents discussed. Christie Davidson, Yezdi Lashkari and Henry Lieberman helped in the preparation of the article. ◘

About the Author:

PATTIE MAES is an assistant professor at the MIT Media Laboratory. Her research interests are in the areas of Artificial Intelligence, Artificial Life and Human-Computer Interaction. In particular, she is interested in modeling and building autonomous agents. **Author's Present Address:** The Media Laboratory, MIT Building E15-305B, 20 Ames St, Cambridge, MA 02139; email: pattie@media.mit.edu.

References

1. Chin, D. Intelligent interfaces as agents. In J. Sullivan and S. Tyler, eds., *Intelligent User Interfaces.* ACM Press, N.Y., 1991.
2. Crowston, K. and Malone, T. Intelligent software agents. *BYTE, 13,* 13 (Dec. 1988), 267–271.
3. Cypher, A. EAGER: Programming repetitive tasks by example. In *Proceedings of CHI'91.* ACM Press, N.Y., 1991, pp. 33–39.
4. Dent, L. Boticario, J. Mc Dermott, J. Mitchell, T. and Zabowski, D. A personal learning apprentice. In *Proceedings of the National Conference on Artificial Intelligence.* MIT Press, Cambridge, Mass., 1992.
5. Don, A. Anthropomorphism: From Eliza to Terminator 2, panel description. In *Proceedings of CHI'92.* ACM Press, N.Y., 1992, pp. 67–70.
6. Dorner, S. Eudora Reference Manual. Qualcomm Inc., 1992.
7. Feynman, C. Nearest neighbor and maximum likelihood methods for social information filtering. Internal report, MIT, Dec. 1993.
8. Kay, A. Computer software. In *Sci. Amer. 251,* 3 (Mar. 1984), 191–207.
9. Kay, A. User Interface: A personal

CONTINUED ON PAGE 146

Acknowledgments

Steven Ketchpel and Chris Ramming implemented various parts of the system. We thank Ron Brachman, Oren Etzioni, Mark Jones, and Eric Sumner for useful suggestions and feedback.

References

1. Dent, L., Boticario, J., McDermott, J., Mitchell, T. and Zabowski, D. A personal learning apprentice. In *Proceedings of AAAI-92*, AAAI Press/The MIT Press, 1992, pp. 96–103.
2. Etzioni, O., Hanks, S., Weld, D., Draper, D., Lesh, N. and Williamson, M. An approach to planning with incomplete information. In *Proceedings of KR-92*, Morgan Kaufmann, 1992, pp. 115–125.
3. Etzioni, O., Lesh, N. and Segal, R. Building softbots for Unix. Tech. Rep., University of Washington, Seattle, Wash., 1992.
4. Maes, P., Ed. *Designing Autonomous Agents*. MIT/Elsevier, 1993.
5. Shoham, Y. Agents-oriented programming. *Artif. Intell. 60* (1993) 51–92.

About the Authors:

HENRY A. KAUTZ is a research scientist at AT&T Bell Laboratories. Research interests cover various topics in artificial intelligence, including fast approximate reasoning algorithms, planning, and software agents.

BART SELMAN is a research scientist at AT&T Bell Laboratories. Interests include randomized algorithms for satisfiability testing, the complexity of commonsense inference, and various topics in artificial intelligence.

Author's Present Address: Kautz and Selman can be reached at AT&T Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974; email {kautz, selman}@research.att.com

MICHAEL COEN is a graduate student at the MIT Artificial Intelligence Laboratory. He is interested in tools that help simplify software agent construction and provide guarantees of behavioral correctness. **Author's Present Address:** MIT Artificial Intelligence Laboratory, 545 Technology Square, NE43-823, Cambridge, MA 02139; email mhcoen@ai.mit.edu

CONTINUED FROM PAGE 40

view. In B. Laurel, ed., *The Art of Human-Computer Interface Design*. Addison-Wesley, Reading, Mass., 1990.
10. Kozierok, R. and Maes, P. A learning interface agent for scheduling meetings. In *Proceedings of ACM SIGCHI International Workshop on Intelligent User Interfaces*. ACM Press, N.Y., 1993, pp. 81–88.
11. Kozierok, R. A learning approach to knowledge acquisition for intelligent interface agents. SM Thesis, Department of Electrical Eng. and Computer Science, MIT, May 1993.
12. Lai, K. Malone, T. and Yu, K. Object Lens: A "spreadsheet" for cooperative work. *ACM Trans. Office Inf. Syst. 6*, 4 (Apr. 1988), 332–353.
13. Lashkari, Y., Metral, M. and Maes, P. Collaborative interface agents. In *Proceedings of the National Conference on Artificial Intelligence*. MIT Press, Cambridge, Mass., 1994.
14. Laurel, B. Interface agents: Metaphors with character. In B. Laurel, ed., *The Art of Human-Computer Interface Design*. Addison-Wesley, Reading, Mass., 1990.
15. Lieberman, H. Mondrian: A teachable graphical editor. In A. Cypher, ed., *Watch what I do: Programming by Demonstration*, MIT Press, Cambridge, Mass., 1993.
16. Maes, P. and Kozierok, R. Learning interface agents. In *Proceedings of the AAAI'93 Conference*. MIT Press, Cambridge, Mass., pp. 459–465.
17. Myers, B. *Creating User Interfaces by Demonstration*. Academic Press, Boston, Mass., 1988.
18. Myers, B., ed. Demonstrational interfaces: Coming soon? In *Proceedings of CHI'91*. ACM Press, N.Y., 1991, pp. 393–396.
19. Negroponte, N. *The Architecture Machine; Towards a more Human Environment*. MIT Press, Cambridge, Mass., 1970.
20. Salton, G. and McGill, M. *Introduction to Modern Information Retrieval*. McGraw-Hill, N.Y., 1983.
21. Schneiderman, B. Direct manipulation: A step beyond programming languages. *IEEE Comput. 16*, 8 (Aug. 1988), 57–69.
22. Sheth, B. and Maes, P. Evolving agents for personalized information filtering. In *Proceedings of the Ninth Conference on Artificial Intelligence for Applications*. IEEE Computer Society Press, 1993.
23. Sheth, B. A learning approach to personalized information filtering. SM Thesis, Department of Electrical Eng. and Computer Science, MIT, Feb. 1994.
24. Stanfill, C. and Waltz, D. Toward memory-based reasoning. *Comm. ACM 29*, 12, (Dec. 1986), 1213–1228.
25. Sullivan, J.W. and Tyler, S.W., eds. *Intelligent User Interfaces*. ACM Press, N.Y., 1991.

This research is sponsored by Apple Computer, by the NSF under grant number IRI-92056688, and by the News in the Future Consortium of the MIT Media Laboratory.

CONTINUED FROM PAGE 116

24. Winston, P.H., Ginford, T.O., Katz, B., and Lowry, M. Learning physical descriptions from functional definitions, examples, and precedents. In *Proceedings National Conference on Artificial Intelligence*, AAAI/MIT Press, Cambridge, Mass., 1983, pp. 433–439.

About the Author:

DOUG RIECKEN is a principal investigator in the Computer Systems Research Laboratory at AT&T Bell Laboratories. Current research interests include artificial intelligence, multimedia technologies, and human-computer interactions. **Author's Present Address:** Computer Systems Research Laboratory, AT&T Bell Laboratories, Room 4F-611, Crawfords Corner Road, Holmdel, NJ 07733-3030; email: wolf@research.att.com